Informe de Laboratorio 12

Tema: Definición de Clases de Usuario Clase Soldado - Menú

Nota

Estudiante	Escuela	Asignatura
Roni Companocca Checco rcompanocca@unsa.edu.pe	Escuela Profesional de Ingeniería de Sistemas	Programación Semestre: II Código: 20210558

Laboratorio	Tema	Duración
12	Definición de Clases de Usuario Clase Soldado - Menú	04 horas

Semestre académico	Fecha de inicio	Fecha de entrega
2023 - B	Del 28 octubre 2023	Al 2 Noviembre 2023

1. TAREA

1.1. Objetivos:

- Que el alumno demuestre poder crear "clases definidas por el programador"
- Implementar métodos para las clases definidas por el programador

1.2. Competencias a alcanzar:

- Diseña, responsablemente, sistemas, componentes o procesos para satisfacer necesidades dentro de restricciones realistas: económicas, medio ambientales, sociales, políticas, éticas, de salud, de seguridad, manufacturación y sostenibilidad.
- Aplica de forma flexible, técnicas, métodos, principios, normas, estándares y herramientas de ingeniería necesarias para la construcción de software e implementación de sistemas de información.

1.3. Marco teorico:

- Clase: Una Fábrica de objetos. Una clase está conformada por dos partes: datos miembro (variables de instancia / atributos) y los métodos.
- Los métodos nos permiten acceder y/o modificar los datos miembros (atributos) de los objetos.
- Toda clase necesita al menos un Constructor. El constructor lleva el mismo nombre de la clase. El constructor es un método especial que siempre se llama con la palabra reservada new()

1.4. Indicaciones generales:

- Todos los ejercicios deberán ser guardados en el mismo Proyecto
- El Proyecto deberá tener el nombre del Laboratorio y el nombre del alumno, así por ejemplo: Laboratorio 1 – Juan Perez
- Cada Clase deberá tener el nombre del ejercicio, así por ejemplo: Ejercicio1
- Utilice nombres de variables significativos y todas las recomendaciones de estilo
- Especialmente, su código deberá estar correctamente indentado
- Deberá pasar TODOS los casos de prueba

2. EQUIPOS, MATERIALES Y TEMAS UTILIZADOS

- Sistema Operativo Windows
- OpenJDK 64-Bits 17.0.7.
- Git 2.39.2.
- Cuenta en GitHub con el correo institucional.

3. URL DE REPOSITORIO GITHUB

- URL para el Repositorio GitHub.
- `https://github.com/RONI-COMPANOCCA-CHECCO`
- URL para el laboratorio 12 en el Repositorio GitHub.
- `https://github.com/RONI-COMPANOCCA-CHECCO/FP2-LAB12`

4. EJERCICIO PROPUESTO

4.1. INTRODUCCION

4.1.1. Este laboratorio requiere que usted escriba un programa utilizando clases definidas por el programador. No deberá utilizar sintaxis o constructores que no han sido cubiertos durante las clases teóricas. Será penalizado por esta falta. A menos que una plantilla sea dada, deberá utilizar cada programa desde cero de manera que obtenga suficiente práctica en la escritura de programas en Java.

4.1.2. Un consejo: Programe incrementalmente. No trate de terminar todas las partes del programa y luego compilarlo. Escriba sus programas en partes y compílelo de forma frecuente. Trate de mantener un programa compilable aun cuando esté trabajando en él. Presentar un programa compilable que funcione parcialmente es mejor que presentar un programa no-compilable. EN SERIO, programe incrementalmente.

4.1.3. Los objetivos de este laboratorio son:

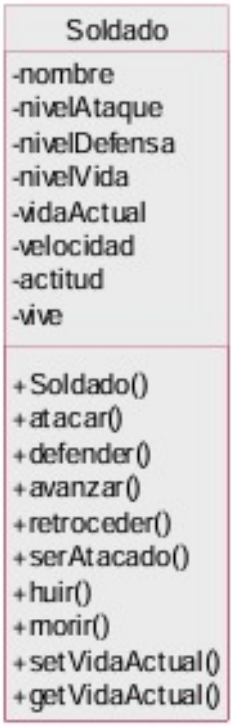
4.1.4. Deberá asumir que todos los datos de ingreso son correctos.

4.1.5. Deberá utilizar la clase Scanner en System.in para ingresos de datos y System.out para salida de datos en sus programas, a menos que se indique lo contrario.

4.1.6. Pruebe sus programas con sus propios datos de prueba antes de presentarlos.

4.1.7. Evitar duplicación de código.

4.1.8. Usar como base el diagrama de clases UML siguiente (puede aumentar atributos y métodos necesarios):



- .

4.1.9. Puede reutilizar todo el código del laboratorio 11, pero ahora el objetivo es gestionar los ejércitos autogenerados.

4.1.10. Al ejecutar el videojuego, el programa deberá dar las opciones:

1. Juego rápido (tal cual como en el laboratorio 11) Al acabar el juego mostrar las opciones de volver a jugar y de volver al menú principal. También se deberá tener la posibilidad de cancelar el juego actual en cualquier momento, permitiendo escoger entre empezar un juego totalmente nuevo o salir al menú principal.

2. Juego personalizado: permite gestionar ejércitos. Primero se generan los 2 ejércitos con sus respectivos soldados y se muestran sus datos. Luego se tendrá que escoger cuál de los 2 ejércitos se va a gestionar, después se mostrarán las siguientes opciones:

2.1 Crear Soldado: permitirá crear un nuevo soldado personalizado y añadir al final del ejército (recordar que límite es de 10 soldados por ejército)

2.2 Eliminar Soldado (no debe permitir un ejército vacío)

2.3 Clonar Soldado (crea una copia exacta del soldado) y se añade al final del ejército (recordar que límite es de 10 soldados por ejército)

2.4 Modificar Soldado (con submenú para cambiar alguno de los atributos nivelAtaque, nivelDefensa, vidaActual)

2.5 Comparar Soldados (verifica si atributos: nombre, nivelAtaque, nivelDefensa, vidaActual y vive son iguales)

2.6 Intercambiar Soldados (intercambia 2 soldados en sus posiciones en la estructura de datos del ejército)

2.7 Ver soldado (Búsqueda por nombre)

2.8 Ver ejército

2.9 Sumar niveles (usando Method-Call Chaining), calcular las sumatorias de nivelVida, nivelAtaque, nivelDefensa, velocidad de todos los soldados de un ejército

2.10 Jugar (se empezará el juego con los cambios realizados) y con las mismas opciones de la opción 1.

2.11 Volver (muestra el menú principal) Después de escoger alguna de las opciones 1) a 9) se podrá volver a elegir uno de los ejércitos y se mostrarán las opciones 1) a 11)

3 . Salir

- la clase Main.java

```
// RONI COMPANOCCA CHECCO
// CUI: 20210558
// LABORATORIO 12
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;

public class Main {

    public static void main(String[] args) {

      //DECLARACION DE VARIABLES Y ARREGLOS NECESARIOS
      ArrayList<Soldado> ejercito1 = new ArrayList();
      ArrayList<Soldado> ejercito2 = new ArrayList();
      ArrayList<ArrayList<Soldado>> tablero = new ArrayList();
      int batallon1, batallon2;
      int vidatotal1=0, vidatotal2=0;
      double promedioVida1=0, promedioVida2=0;

      // BUCLE PARA DESIGNAR LA CANTIDAD DE FILAS Y COLUMNAS DEL TABLERO
      for(int i=0; i<10; i++) {
         tablero.add(new ArrayList<Soldado>());
         for(int j=0; j<10; j++) {
            tablero.get(i).add(new Soldado());
         }
      }

      // CREACION DEL NUMERO DE POSICIONES DE CADA EJERCITO
      batallon1 = aleatorio(1,10);
      batallon2 = aleatorio(1,10);

      // INICIALIZAR ARREGLOS
      inicializarArreglo(ejercito1, batallon1);
      inicializarArreglo(ejercito2, batallon2);

      // GENERAR EJERCITOS VALIDOS
      generarEjercitos(ejercito1, ejercito2);

      //  AADIR  LOS EJERCITOS AL TABLERO
      aadirTablero(ejercito1, tablero);
      aadirTablero(ejercito2, tablero);

      //IMPRIMIR EL TABLERO
      imprimirTablero(tablero);

      //IMPRIMIR LOS SOLDADOS DE MAYOR VIDA DE CADA EJERCITO
      System.out.println("Soldado de mayor vida del ejercito 1");
      SoldadoConMayorVida(ejercito1);
      System.out.println("soldado de mayor vida del ejercito 2");
      SoldadoConMayorVida(ejercito2);

      //IMPRIMIR LA VIDA TOTAL Y EL PROMEDIO DEL EJERCITO 1
      System.out.println("\nEJERCITO 1: ");
```

```
    for (int i=0; i<ejercito1.size(); i++) {
       vidatotal1+=ejercito1.get(i).getPuntos();
       promedioVida1 = vidatotal1/(ejercito1.size()*1.0);
    }
    System.out.println("Vida total: "+vidatotal1);
    System.out.println("Promedio de vida: "+promedioVida1);

    //IMPRIMIR LA VIDA TOTAL Y EL PROMEDIO DEL EJERCITO 2
    System.out.println("\nEJERCITO 2: ");
    for (int i=0; i<ejercito2.size(); i++) {
       vidatotal2+=ejercito2.get(i).getPuntos();
       promedioVida2 = vidatotal2/(ejercito2.size()*1.0);
    }
    System.out.println("Vida total: "+vidatotal2);
    System.out.println("Promedio de vida: "+promedioVida2);

    //IMPRIMIR LOS SOLDADOS CREADOS EN EL ORDEN POR DEFECTO
    System.out.println("\nLista ejercito 1:");
    for(int i=0; i<ejercito1.size(); i++) {
       imprimir(ejercito1.get(i));
    }
    System.out.println("\nLista ejercito 2:");
    for(int i=0; i<ejercito2.size(); i++) {
       imprimir(ejercito2.get(i));
    }

    // IMPRIMIR LOS DATOS DE LOS SOLDADOS ORDENADOS DE MAYOR A MENOR DEPENDIENDO DE SU
        NIVEL DE VIDA USANDO DOS TIPOS DE ALGORITMO
    ordenarPorVidaMetodoA(ejercito1);
    ordenarPorVidaMetodoB(ejercito2);
    System.out.println("\nEjercito 1 Ordenados por nivel de vida");
    for(int i=0; i<ejercito1.size(); i++) {
       imprimir(ejercito1.get(i));
    }
    System.out.println("\nEjercito 2 Ordenados por nivel de vida");
    for(int i=0; i<ejercito2.size(); i++) {
       imprimir(ejercito2.get(i));
    }

    // MOSTRAR EJERCITO GANADOR LA METRICA USADA ARA DESIGNAR AL GANADOR ES POR EL
        NIVEL DEL PROMEDIO DE VIDA DE CADA EJERCITO
    if(promedioVida1>promedioVida2) {
       System.out.println("\nGANADOR ***EJERCITO 1***");
    }else if (promedioVida1<promedioVida2) {
       System.out.println("\nGANADOR ***EJERCITO 2***");
       }
    else {
       System.out.print("\n***ES UN EMPATE***");
    }
 }

  // METODO PARA CREAR NUMEROS ALEATORIOS EN UN RANGO
  public static int aleatorio(int min, int max) {
      return (int) (Math.random() * (max - min + 1) + min);
  }
```

```
// METODO PARA INICIAR UN ARRAYLIST
public static void inicializarArreglo(ArrayList<Soldado> soldadito, int num) {
    for (int i = 0; i < num; i++) {
        soldadito.add(new Soldado());
    }
}

// METODO PARA GENERAR DATOS DEL OBJETO SOLDADO
public static Soldado generarDatos() {
    Soldado soldadito = new Soldado();
    soldadito.setNivelVida(aleatorio(1, 5));
    soldadito.setFila(aleatorio(1, 10));
    soldadito.setColumna(aleatorio(1, 10));
    return soldadito;
}

// METODOS PARA GENERAR LOS EJERCITOS DE MANERA ALEATORIA
public static void generarEjercitos(ArrayList<Soldado> B1, ArrayList<Soldado> B2) {
    ArrayList<Soldado> Soldados = new ArrayList<>();
    Soldados.add(generarDatos());
    for (int i = 1; i < (B1.size() + B2.size()); i++) {
        Soldados.add(generarDatos());
        for (int j = 0; j < i; j++) {
            if (Soldados.get(i).getFila() == Soldados.get(j).getFila()) {
                if (Soldados.get(i).getColumna() == Soldados.get(j).getColumna()) {
                    Soldados.remove(i);
                    i--;
                }
            }
        }
    }
    for (int i = 0; i < B1.size(); i++) {
        B1.add(i, Soldados.get(i));
        B1.get(i).setNombre("Soldado" + i + "x1");
        B1.get(i).setActitud(B1.get(i).getNivelVida() + "[E1]");
        B1.remove(i + 1);
    }
    for (int i = 0; i < B2.size(); i++) {
        B2.add(i, Soldados.get(i + B1.size()));
        B2.get(i).setNombre("Soldado" + i + "x2");
        B2.remove(i + 1);
        B2.get(i).setActitud(B2.get(i).getNivelVida() + "[E2]");
    }
}

// METODO PARA AADIR LOS EJERCITOS AL TABLERO
public static void aadirTablero(ArrayList<Soldado> soldadito,
     ArrayList<ArrayList<Soldado>> table) {
    for (int i = 0; i < soldadito.size(); i++) {
        table.get(soldadito.get(i).getColumna() - 1).add(soldadito.get(i).getFila()
            - 1, soldadito.get(i));
        table.get(soldadito.get(i).getColumna() -
            1).remove(soldadito.get(i).getFila());
    }
}
```

```
// METODO PARA IMPRIMIR EL TABLERO EN LA CUAL SE DESARROLLA EL JUEGO
public static void imprimirTablero(ArrayList<ArrayList<Soldado>> table) {
    System.out.println("\tA\tB\tC\tD\tF\tG\tH\tI\tJ");
    for (int i = 0; i < table.size(); i++) {
        System.out.print(i + 1);
        for (int j = 0; j < table.get(i).size(); j++) {
            System.out.print("\t" + table.get(i).get(j).getActitud());
        }
        System.out.println("\n");
    }
}

// METODO PARA IMPRIMIR LOS SOLDADOS DE MAYOR VIDA
public static void SoldadoConMayorVida(ArrayList<Soldado> soldadito) {
    Soldado mayor = new Soldado();
    mayor.setNivelVida(0);
    for (int i = 0; i < soldadito.size(); i++) {
        if (mayor.getNivelVida() < soldadito.get(i).getNivelVida()) {
            mayor = soldadito.get(i);
        }
    }
    imprimir(mayor);
}

// METODO PARA IMPRIMIR EL NOMBRE, LA POSICION Y NIVEL DE VIDA DEL SOLDADO
public static void imprimir(Soldado soldadito) {
    System.out.println("Nombre: " + soldadito.getNombre() + "\nPosicion: " +
        soldadito.getColumna() + "X" + soldadito.getFila() + "\tVida: " +
        soldadito.getNivelVida());
}

// METODO QUE NOS AYUDA A ORDENAR LOS SOLDADOS DE ACUERDO A SU NIVEL DE VIDA,
    USUANDO UN ALGORITMO DE ORDENAMIENTO DE BURBUJA
public static void ordenarPorVidaMetodoA(ArrayList<Soldado> soldadito) {
    Soldado aux = new Soldado();
    for (int i = 0; i < soldadito.size() - 1; i++) {
        for (int j = 0; j < soldadito.size() - i - 1; j++) {
            if (soldadito.get(j).getNivelVida() < soldadito.get(j +
                1).getNivelVida()) {
                aux = soldadito.get(j);
                soldadito.set(j, soldadito.get(j + 1));
                soldadito.set(j + 1, aux);
            }
        }
    }
}

// METODO QUE NOS AYUDA A ORDENAR LOS SOLDADOS DE ACUERDO A SU NIVEL DE VIDA, EN
    ESTA OCACION DIFERENTE A LA ANTERIOR QUE ERA ALGORITMO DE BURBUJA
public static void ordenarPorVidaMetodoB(ArrayList<Soldado> soldadito) {
    Collections.sort(soldadito, new Comparator<Soldado>() {
        public int compare(Soldado s1, Soldado s2) {
            // Orden descendente por nivel de vida
            return Integer.compare(s2.getNivelVida(), s1.getNivelVida());
        }
    });
```

```
    }
}
    // METODO PARA IMPRIMIR EL TABLERO EN LA CUAL SE DESARROLLA EL JUEGO
    public static void imprimirTablero(ArrayList<ArrayList<Soldado>> table) {
        System.out.println("\tA\tB\tC\tD\tF\tG\tH\tI\tJ");
        for (int i = 0; i < table.size(); i++) {
            System.out.print(i + 1);
            for (int j = 0; j < table.get(i).size(); j++) {
                System.out.print("\t" + table.get(i).get(j).getActitud());
            }
            System.out.println("\n");
        }
    }

    // METODO PARA IMPRIMIR LOS SOLDADOS DE MAYOR VIDA
    public static void SoldadoConMayorVida(ArrayList<Soldado> soldadito) {
        Soldado mayor = new Soldado();
        mayor.setNivelVida(0);
        for (int i = 0; i < soldadito.size(); i++) {
            if (mayor.getNivelVida() < soldadito.get(i).getNivelVida()) {
                mayor = soldadito.get(i);
            }
        }
        imprimir(mayor);
    }

    // METODO PARA IMPRIMIR EL NOMBRE, LA POSICION Y NIVEL DE VIDA DEL SOLDADO
    public static void imprimir(Soldado soldadito) {
        System.out.println("Nombre: " + soldadito.getNombre() + "\nPosicion: " +
            soldadito.getColumna() + "X" + soldadito.getFila() + "\tVida: " +
            soldadito.getNivelVida());
    }

    // METODO QUE NOS AYUDA A ORDENAR LOS SOLDADOS DE ACUERDO A SU NIVEL DE VIDA,
        USUANDO UN ALGORITMO DE ORDENAMIENTO DE BURBUJA
    public static void ordenarPorVidaMetodoA(ArrayList<Soldado> soldadito) {
        Soldado aux = new Soldado();
        for (int i = 0; i < soldadito.size() - 1; i++) {
            for (int j = 0; j < soldadito.size() - i - 1; j++) {
                if (soldadito.get(j).getNivelVida() < soldadito.get(j +
                    1).getNivelVida()) {
                    aux = soldadito.get(j);
                    soldadito.set(j, soldadito.get(j + 1));
                    soldadito.set(j + 1, aux);
                }
            }
        }
    }

    // METODO QUE NOS AYUDA A ORDENAR LOS SOLDADOS DE ACUERDO A SU NIVEL DE VIDA, EN
        ESTA OCACION DIFERENTE A LA ANTERIOR QUE ERA ALGORITMO DE BURBUJA
    public static void ordenarPorVidaMetodoB(ArrayList<Soldado> soldadito) {
        Collections.sort(soldadito, new Comparator<Soldado>() {
            public int compare(Soldado s1, Soldado s2) {
                // Orden descendente por nivel de vida
                return Integer.compare(s2.getNivelVida(), s1.getNivelVida());
```

```
            }
        });
    }
}
```

- la clase Soldado.java

```
// RONI COMPANOCCA CHECCO
// CUI: 20210558
// LABORATORIO 12
// FUNDAMENTOS DE PROGRAMACION
// CLASE SOLDADO PARA LOS METODOS SETTER Y GETTER
public class Soldado {
    private String nombre;
    private int nivelAtaque;
    private int nivelDefensa;
    private int nivelVida;
    private int vidaActual;
    private int velocidad;
    private String actitud;
    private boolean vive;

    public Soldado() {
        nombre = "";
        nivelAtaque = 0;
        nivelDefensa = 0;
        nivelVida = 0;
        vidaActual = nivelVida;
        velocidad = 0;
        actitud = "";
        vive = true;
    }

    public void atacar(Soldado enemigo) {
        int danio = this.nivelAtaque - enemigo.nivelDefensa;
        if (danio > 0) {
            enemigo.serAtacado(danio);
            System.out.println(this.nombre + " atac a " + enemigo.nombre + " y le caus "
                + danio + " de dao.");
        } else {
            System.out.println(this.nombre + " atac a " + enemigo.nombre + " pero no le
                caus dao.");
        }
    }

    public void defender() {
        // Lgica para la defensa
        // Puede disminuir el dao recibido en un futuro ataque, por ejemplo
        this.nivelDefensa += 10; // Aumentar la defensa por ejemplo
        System.out.println(this.nombre + " se est defendiendo.");
    }

    public void avanzar() {
        // Lgica para avanzar en el juego
        // Podra mover al soldado a una nueva posicin en el tablero, por ejemplo
        if (this.velocidad > 0) {
```

```
        // Mover al soldado en la direccin correspondiente
        System.out.println(this.nombre + " est avanzando.");
    } else {
        System.out.println(this.nombre + " no puede avanzar, la velocidad es 0.");
    }
}

public void retroceder() {
    // Lgica  para retroceder en el juego
    // Podra  mover al soldado hacia atrs en el tablero, por ejemplo
    System.out.println(this.nombre + " est retrocediendo.");
}

public void serAtacado(int danioRecibido) {
    this.vidaActual -= danioRecibido;
    if (this.vidaActual <= 0) {
        this.morir();
    }
}

public void huir() {
    // Lgica  para huir del combate
    // Podra  cambiar la posicin del soldado a una zona segura, por ejemplo
    if (this.nivelVida < 10) {
        // Huir solo si la vida es baja
        System.out.println(this.nombre + " est huyendo del combate.");
    } else {
        System.out.println(this.nombre + " no puede huir, su vida est alta.");
    }
}

public void morir() {
    this.vidaActual = 0;
    this.vive = false;
}

public void setVidaActual(int vidaActual) {
    this.vidaActual = vidaActual;
}

public int getVidaActual() {
    return vidaActual;
}

// Getters y Setters

public String getNombre() {
    return nombre;
}

public String getNivelVida() {
    return nivelVida;
}

public void setNombre(String nombre) {
    this.nombre = nombre;
```

```
    }

    public int getNivelAtaque() {
        return nivelAtaque;
    }

    public int setNivelVida() {
        this.nivelVida = nivelVida;
    }

    public void setNivelAtaque(int nivelAtaque) {
        this.nivelAtaque = nivelAtaque;
    }

    public int getNivelDefensa() {
        return nivelDefensa;
    }

    public void setNivelDefensa(int nivelDefensa) {
        this.nivelDefensa = nivelDefensa;
    }

    public boolean isVive() {
        return vive;
    }

    public void setVive(boolean vive) {
        this.vive = vive;
    }
}
```

- Ejecucion

```
          A      B      C      D      F      G      H      I      J
1                                          2[E1]
2                                                        5[E1]
3
4
5
6                                                                 2[E2]
7
8                            2[E1]  3[E2]
9      3[E2]                                                      2[E1]
10                                         2[E1]          3[E2]

Soldado de mayor vida del ejercito 1
Nombre: Soldado4x1
```

```
Posicion: 2X8 Vida: 5
soldado de mayor vida del ejercito 2
Nombre: Soldado0x2
Posicion: 10X8 Vida: 3

EJERCITO 1:
Vida total: 13
Promedio de vida: 2.6

EJERCITO 2:
Vida total: 11
Promedio de vida: 2.75

Lista ejercito 1:
Nombre: Soldado0x1
Posicion: 1X6 Vida: 2
Nombre: Soldado1x1
Posicion: 9X9 Vida: 2
Nombre: Soldado2x1
Posicion: 8X4 Vida: 2
Nombre: Soldado3x1
Posicion: 10X6 Vida: 2
Nombre: Soldado4x1
Posicion: 2X8 Vida: 5

Lista ejercito 2:
Nombre: Soldado0x2
Posicion: 10X8 Vida: 3
Nombre: Soldado1x2
Posicion: 8X5 Vida: 3
Nombre: Soldado2x2
Posicion: 6X9 Vida: 2
Nombre: Soldado3x2
Posicion: 9X1 Vida: 3

Ejercito 1 Ordenados por nivel de vida
Nombre: Soldado4x1
Posicion: 2X8 Vida: 5
Nombre: Soldado0x1
Posicion: 1X6 Vida: 2
Nombre: Soldado1x1
Posicion: 9X9 Vida: 2
Nombre: Soldado2x1
Posicion: 8X4 Vida: 2
Nombre: Soldado3x1
Posicion: 10X6 Vida: 2

Ejercito 2 Ordenados por nivel de vida
Nombre: Soldado0x2
Posicion: 10X8 Vida: 3
Nombre: Soldado1x2
Posicion: 8X5 Vida: 3
Nombre: Soldado3x2
Posicion: 9X1 Vida: 3
Nombre: Soldado2x2
Posicion: 6X9 Vida: 2
```

```
GANADOR ***EJERCITO 2***
```

5. REFERENCIAS

- M. Aedo, "Fundamentos de Programación 2 - Tópicos de Programación Orientada a Objetos", Primera Edición, 2021, Editorial UNSA.
- `https://github.com/rescobedoq/programacion.git`
- J. Dean, Ïntroduction to programming with Java: A Problem Solving Approach", Third Edition, 2021, McGraw-Hill.
- C. T. Wu, .ªn Introduction to Object-Oriented Programming with Java", Fifth Edition, 2010, McGraw-Hill.
- P. Deitel, "Java How to Program", Eleventh Edition, 2017, Prentice Hall.